TEST DRIVEN DEVELOPMENT

CHEATSHEET

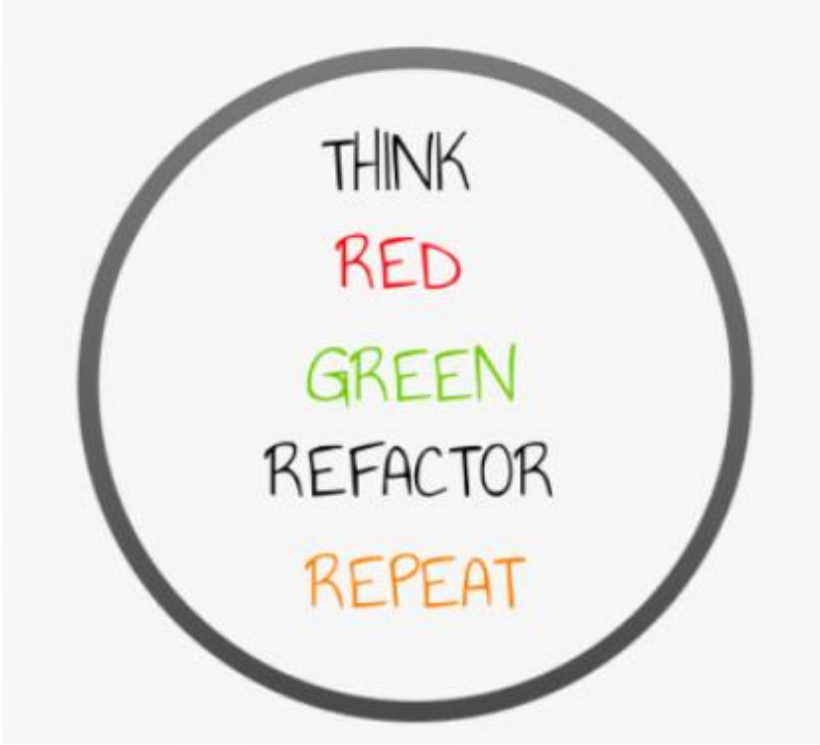
TRAINER: OVIDIU DRUMIA

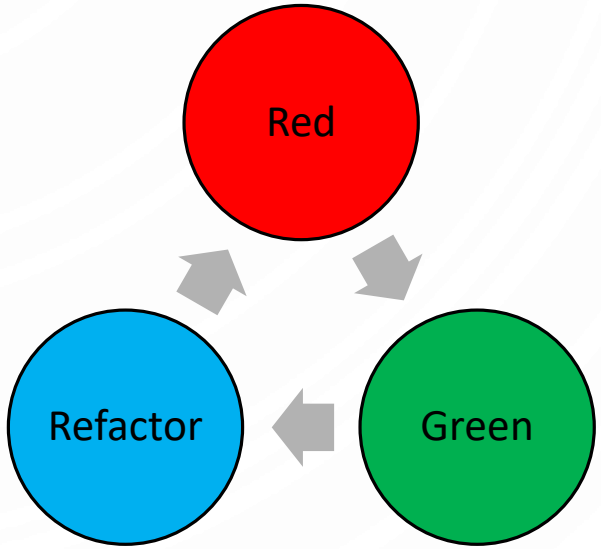
AXP Academy

3 LAWS OF TDD (UNCLE BOB)

- You are not allowed to write any production code unless it is to make a failing unit test pass.
- You are not allowed to write any more of a unit test than is sufficient to fail; and compilation failures are failures.
- You are not allowed to write any more production code than is sufficient to pass the one failing unit test.

ROUTINE





AXP Academy

ROUTINE

THINK

Think about what tests you're writing.
Don't write empty tests.

TAKE YOUR TIME

RED

Write a very small amount of code.
This should break your build.
(1 minute)

GREEN

Write only enough code to fix the test.
Don't worry about it being pretty.
(30 seconds)

REFACTOR

Refactor your code without fear.
Improve the look, remove smells.
After each change, run your test and make sure it still passes.

REPEAT

Do the whole cycle again.
You should be repeating this cycle many times an hour.
(20 - 40)

AXP Academy

WHAT DEFINES GOOD TESTS?

- Fast: tests should run fast!
- Independent: they should not depend on each other
- Repeatable: they should be repeatable in any environment
- Self-validating: RED or GREEN
- Timely: write test before production code

but above all

READABLE

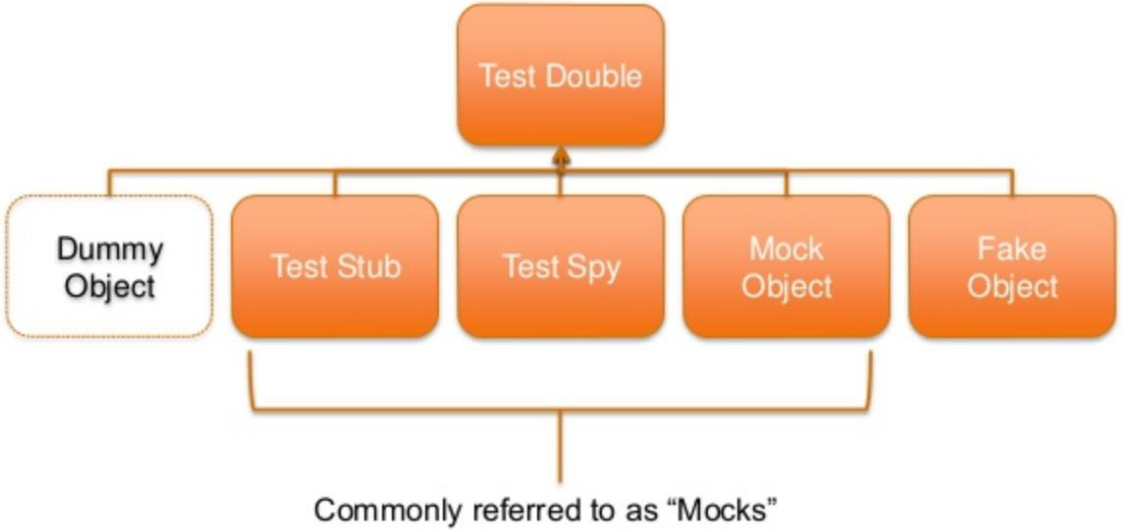
AXP Academy

TDD MISTAKES

1. No test at the beginning
   a) Skipping writing test first “sometimes” when it’s “convenient”
   b) Skipping writing test first mentally (I know exactly what my next 20 steps are)
2. Skipping RED phase
   a) Failing test does not compile
   b) Test has not been run before creating the implementation that satisfies it
   c) Test is broken (it fails because of a different reason than it has been designed)
3. Skipping the Refactor phase
   a) “code should read like a well-written novel” (Uncle Bob)
   b) self-commented: “comments are always failures” (Uncle Bob)
4. Too large steps
   a) Difficult initialization (large “given” section)
   b) Too many assertions (large “then” section)
5. Goal: Code Coverage
   a) Do not create tests to gain 100% coverage
   b) High coverage does not guarantee the functionality is well tested

TEST DOUBLES

- Test Double is a generic term for any case where you replace a production object for testing purposes.



AXP Academy

IN PRACTICE - MOCKITO

MOCKITO – INITIALIZING TEST

```
@RunWith(MockitoJUnitRunner.class)
public class MockitoAnnotationTest {
    ...
}
```

```
@Before
public void init() {
    MockitoAnnotations.initMocks(this);
}
```

MOCKITO – SETTING UP A MOCK

```java
@Test
public void whenNotUseMockAnnotation_thenCorrect() {
    List mockList = Mockito.mock(ArrayList.class);

    mockList.add("one");
    Mockito.verify(mockList).add("one");
    assertEquals(0, mockList.size());

    Mockito.when(mockList.size()).thenReturn(100);
    assertEquals(100, mockList.size());
}
```

```java
@Mock
List<String> mockedList;

@Test
public void whenUseMockAnnotation_thenMockIsInjected() {
    mockedList.add("one");
    Mockito.verify(mockedList).add("one");
    assertEquals(0, mockedList.size());

    Mockito.when(mockedList.size()).thenReturn(100);
    assertEquals(100, mockedList.size());
}
```

MOCKITO – SETTING UP A SPY

```java
@Test
public void whenNotUseSpyAnnotation_thenCorrect() {
    List<String> spyList = Mockito.spy(new ArrayList<String>());

    spyList.add("one");
    spyList.add("two");

    Mockito.verify(spyList).add("one");
    Mockito.verify(spyList).add("two");

    assertEquals(2, spyList.size());

    Mockito.doReturn(100).when(spyList).size();
    assertEquals(100, spyList.size());
}
```

```java
@Spy
List<String> spiedList = new ArrayList<String>();

@Test
public void whenUseSpyAnnotation_thenSpyIsInjected() {
    spiedList.add("one");
    spiedList.add("two");

    Mockito.verify(spiedList).add("one");
    Mockito.verify(spiedList).add("two");

    assertEquals(2, spiedList.size());

    Mockito.doReturn(100).when(spiedList).size();
    assertEquals(100, spiedList.size());
}
```

MOCKITO – SETTING UP A SPY

```java
@Test
public void whenNotUseSpyAnnotation_thenCorrect() {
    List<String> spyList = Mockito.spy(new ArrayList<String>());

    spyList.add("one");
    spyList.add("two");

    Mockito.verify(spyList).add("one");
    Mockito.verify(spyList).add("two");

    assertEquals(2, spyList.size());

    Mockito.doReturn(100).when(spyList).size();
    assertEquals(100, spyList.size());
}
```

```java
@Spy
List<String> spiedList = new ArrayList<String>();

@Test
public void whenUseSpyAnnotation_thenSpyIsInjected() {
    spiedList.add("one");
    spiedList.add("two");

    Mockito.verify(spiedList).add("one");
    Mockito.verify(spiedList).add("two");

    assertEquals(2, spiedList.size());

    Mockito.doReturn(100).when(spiedList).size();
    assertEquals(100, spiedList.size());
}
```

MOCKITO – CAPTOR

```java
@Test
public void whenNotUseCaptorAnnotation_thenCorrect() {
    List mockList = Mockito.mock(List.class);
    ArgumentCaptor<String> arg = ArgumentCaptor.forClass(String.class);

    mockList.add("one");
    Mockito.verify(mockList).add(arg.capture());

    assertEquals("one", arg.getValue());
}
```

```java
ArgumentCaptor<Person> argument = ArgumentCaptor.forClass(Person.class);
verify(mock).doSomething(argument.capture());
assertEquals("John", argument.getValue().getName());
```

```java
@Mock
List mockedList;

@Captor
ArgumentCaptor argCaptor;

@Test
public void whenUseCaptorAnnotation_thenTheSam() {
    mockedList.add("one");
    Mockito.verify(mockedList).add(argCaptor.capture());

    assertEquals("one", argCaptor.getValue());
}
```


AXP Academy

MOCKITO – INJECT MOCKS

```java
@Mock
Map<String, String> wordMap;

@InjectMocks
MyDictionary dic = new MyDictionary();

@Test
public void whenUseInjectMocksAnnotation_thenCorrect() {
    Mockito.when(wordMap.get("aWord")).thenReturn("aMeaning");

    assertEquals("aMeaning", dic.getMeaning("aWord"));
}
```

```java
public class MyDictionary {
    Map<String, String> wordMap;

    public MyDictionary() {
        wordMap = new HashMap<String, String>();
    }
    public void add(final String word, final String meaning) {
        wordMap.put(word, meaning);
    }
    public String getMeaning(final String word) {
        return wordMap.get(word);
    }
}
```


AXP Academy

MOCKITO – NPE USING MOCKS

```
1  public class NPETest {
2
3      @Mock
4      List mockedList;
5
6      @Test
7      public void test() {
8          Mockito.when(mockedList.size()).thenReturn(1);
9      }
10 }
```

- Same behavior using @Spy

LABS – DAY 1

AXP Academy

EXERCISE WARM UP – SPACEBOOK

(HTTPS://GITHUB.COM/OVIDIUDRUMIA/SPACEBOOK)

- Scenario 1: Person has a username
- Scenario 2: Person has a gender
- Scenario 3: Person has an age
- Scenario 4: Person has a list of friends

AXP Academy

EXERCISE – SPACEBOOK

(HTTPS://GITHUB.COM/OVIDIUDRUMIA/SPACEBOOK)

- User Story 1:
  - A **person** has a **username** and a list of **friends**.
  - Username cannot be null, empty or spaces
- User Story 2:
  - Becoming a friend means adding a bidirectional relationship.
  - Adding yourself should not be possible
- User Story 3:
  - A person can **receive messages** from a friend only.
  - A message has a date, sender and body (text)
  - You can ask a person for all received messages
- User Story 4:
  - You can ask a person for all received messages, **sorted** by date.
- User Story 5:
  - You can ask a person for received messages from a friend, sorted by date.

AXP Academy

LABS – DAY 2

AXP Academy

BUG FIX – DRACULA

HTTPS://GITHUB.COM/OVIDIUDRUMIA/DRACULA

- User Story:
  - A hunter can go hunting vampires between midnight and 6 am
- Bug:
  - Hunter can go hunting on midnight
  - EXPECTED: Can go hunting
  - ACTUAL: Cannot go hunting

AXP Academy

LAB 3 – PET SHOP

HTTPS://GITHUB.COM/OVIDIUDRUMIA/PETSHOP

- User Story 1:
  - A PetShop has a stock. Items can be added
- User Story 2:
  - When a new item is added, an email is sent with the name of the item

AXP Academy

LAB 4 – FLIGHT

HTTPS://GITHUB.COM/OVIDIUDRUMIA/FLIGHT/

- User Story 1:
  - A Flight has a list of Passengers
  - A Flight has an Id (int)
  - Passengers only have a name
- User Story 2:
  - You can add Passengers to a Flight
  - You can get the Number of Passengers on a Flight
  - You can check if a Passenger is on a Flight
- User Story 3:
  - A flight has a Maximum Number of Seats
  - When the flight is full, adding a Passenger causes an exception
- User Story 4:
  - You can use a FlightBookingService to book a seat on a Flight using a FlightId and a person
  - Booking a Seat means adding the Passenger to the Flight
  - If the Flight with the given Id is not found an exception is thrown
  - A Flight is loaded from a database (customer is not sure of which DB to use)

AXP Academy

LAB 5 – MOVIE RENTAL

HTTPS://GITHUB.COM/OVIDIUDRUMIA/MOVIE

- Refactor the code using the catalog (http://refactoring.com/catalog/)
- User Story 1:
- Add a new type of movie (Adult)
- Price is € 5 per day
- Think SOLID
  - S – SRP – Single Reponsibility Principle
  - O – OCP – Open/Closed Principle
  - L – LSP – Liskov Substitution Principle
  - I - ISP - Interface Segregation Principle
  - D – DSP – Depencency Inversion Principle